# Fun with Processes (Due Mar. 6, 2018)

Do every problem by yourself. **Do NOT** search for solutions from internet, previous classes or other students. Read Sec. 2.1.7 before you start the problems.

## Problem 1 (10 pts)

A computer has 4 GB of RAM of which the operating system occupies 512 MB. The processes are all 256 MB (for simplicity) and have the same characteristics. If the goal is 99% CPU utilization, what is the maximum I/O wait that can be tolerated?

### Answer

Available RAM = 4096 MB - 512MB = 3584 MB  
3584 / 256 = 14 simultaneous processes  
Maximum I/O wait that can be tolerated...  
1 - p^14 = 0.99  
1 - 0.99 = p^14  
root14(1 - 0.99) = root14(p^14)  
71.97% = p

## Problem 2 (10 pts)

Multiple jobs can run in parallel and finish faster than if they had run sequentially. Suppose that two jobs, each needing 20 minutes of CPU time, start simultaneously. How long will the last one take to complete if they run sequentially? How long if they run in parallel? Assume 50% I/O wait.

### Answer

Time if sequential...  
I/O wait = 50%, CPU time = 20 mins, Total job time = 40 mins each  
  
Total time to complete if sequential = 80 mins  
  
---  
  
Time if parallel...  
1 - 0.5^2 = 0.75 / 2 = 0.375 CPU Utilization / min  
20 min CPU time = 20 / 0.375 = 53.3 mins  
  
Total time to complete if parallel = 53.3 mins

## Problem 3 (10 pts)

Write a simple test program to find out the *parent process id* of a process **before** and **after** its parent process terminates. Based on your test results, answer the following question: what is a process's parent process after its original parent process terminates?

### Answer

$ make run  
Parent process ID while running is: 21961  
# Main process exits  
dsifford ~/.../CSC\_4420/assignment\_02/problem\_03 (master)  
# printf from childprocess still executing  
Parent process ID after exit is: 1

What is a process's parent process after its original process terminates: PID = 1

## Problem 4 (10 pts)

Write a simple test program to find out PIDs of ALL ancestor (i.e., parent, grandparent) processes of the current process or a given process specified by a PID. A parent process's PID of a given process pid can be found in /proc/\$pid/status

### Answer

See problem\_04/ancestors.sh

## Problem 5 (20 pts)

(For those who doesn't have this edition of the textbook, the problem is as follows: If you have a personal UNIX-like system available that you can safely crash and reboot, write a shell script that attempts to create an unlimited number of child processes and observe what happens. Before running the experiment, type sync to the shell to flush the file system buffers to disk to avoid ruining the file system.)

Again, do **NOT** run the experiment on any shared system. You can write a C/C++ program instead of shell script for this experiment. Do **NOT** search for solutions. (There is one on internet, but I doubt you would understand it, otherwise you wouldn't need to take this course now.)

Run the program on your **OWN** laptop/desktop, and record the maximum number of child processes you see before your system crashes, as well as your machine's memory size.

### Answer

See problem\_05/fork\_bomb.sh (output from Docker container in problem\_05/README.md)

## Problem 6 (20 pts)

Design and run an experiment to measure the CPU time used by *fork*. (You may want to use the system call *times*.) Also report the CPU of the machine you run the experiment.

### Answer

See problem\_06/main.c

#### /proc/cpuinfo

processor : 1  
vendor\_id : GenuineIntel  
cpu family : 6  
model : 158  
model name : Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz  
stepping : 9  
microcode : 0x5e  
cpu MHz : 801.352  
cache size : 8192 KB  
physical id : 0  
siblings : 8  
core id : 1  
cpu cores : 4  
apicid : 2  
initial apicid : 2